



# Red Hat Ansible Automation Platform 2.0-ea

## Ansible Builder Guide

Execution environment builder to create consistent and reproducible automation execution environments for your Red Hat Ansible Automation Platform.



# Red Hat Ansible Automation Platform 2.0-ea Ansible Builder Guide

---

Execution environment builder to create consistent and reproducible automation execution environments for your Red Hat Ansible Automation Platform.

## Legal Notice

Copyright © 2021 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java<sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

XFS<sup>®</sup> is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js<sup>®</sup> is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack<sup>®</sup> Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## Abstract

Providing Feedback: If you have a suggestion to improve this documentation, or find an error, create an issue at <http://issues.redhat.com>. Select the Ansible Automation Platform project and use the Documentation component.

---

## Table of Contents

<b>PREFACE</b> .....	<b>3</b>
<b>CHAPTER 1. INTRODUCTION TO ANSIBLE BUILDER</b> .....	<b>4</b>
1.1. ABOUT ANSIBLE BUILDER	4
1.1.1. Why use Ansible Builder?	4
<b>CHAPTER 2. USING ANSIBLE BUILDER</b> .....	<b>5</b>
2.1. INSTALLING ANSIBLE BUILDER	5
2.2. BUILDING A DEFINITION FILE	5
2.3. EXECUTING THE BUILD AND CREATING COMMANDS	6
2.4. BREAKDOWN OF DEFINITION FILE CONTENT	6
2.4.1. Build args and base image	6
2.4.2. Ansible config file path	7
2.4.3. Dependencies	7
2.4.3.1. Galaxy	7
2.4.3.2. Python	8
2.4.3.3. System	8
2.4.4. Additional custom build steps	8
2.5. OPTIONAL BUILD COMMAND ARGUMENTS	9
2.6. CREATING A CONTAINERFILE WITHOUT BUILDING AN IMAGE	9
<b>CHAPTER 3. PUBLISHING AN AUTOMATION EXECUTION ENVIRONMENT</b> .....	<b>10</b>
3.1. PUSHING AN EXECUTION ENVIRONMENT CONTAINER IMAGE TO AUTOMATION HUB	10
3.2. PULLING FROM A PROTECTED REGISTRY	10
<b>CHAPTER 4. BUILDING OFF OF EXISTING BASE EES PROVIDED BY RED HAT ANSIBLE AUTOMATION PLATFORM</b> .....	<b>11</b>
4.1. GATHERING SYSTEM-LEVEL DEPENDENCIES	11
4.2. NOTE FOR PIP-BASED REQUIREMENTS	11
4.3. CUSTOMIZING AN EXISTING EXECUTION ENVIRONMENT IMAGE	11



# PREFACE

Use Ansible Builder to create consistent and reproducible automation execution environments for your Red Hat Ansible Automation Platform needs.

# CHAPTER 1. INTRODUCTION TO ANSIBLE BUILDER

## 1.1. ABOUT ANSIBLE BUILDER

Ansible Builder is a command line tool that automates the process of building automation execution environments by using the metadata defined in various Ansible Collections, as well as by the user.

### 1.1.1. Why use Ansible Builder?

Before Ansible Builder was developed, Automation Platform users would potentially run against dependency issues and multiple error messages as they attempted to create a custom virtual environment or container that had all of their required dependencies installed.

Through the use of an easily customizable definition file, Ansible Builder installs Ansible, specified Collections and any of its dependencies so that all of the necessary requirements to get jobs running are fulfilled behind the scenes.



## CHAPTER 2. USING ANSIBLE BUILDER

### 2.1. INSTALLING ANSIBLE BUILDER

You can install Ansible Builder using Red Hat Subscription Management to register and attach to your Red Hat Ansible Automation Platform subscription. In your terminal, run the following command:

```
$ dnf install ansible-builder
```

You can also install Ansible Builder from the [Python Package Index \(PyPI\)](#). To install with this setup, run:

```
$ pip install ansible-builder
```

### 2.2. BUILDING A DEFINITION FILE

Once you have Ansible Builder installed, we will need to create a definition file which Ansible Builder will use to create your automation execution environment image. The high level process to build an automation execution environment image is for Ansible Builder to read and validate your definition file, then create a **Containerfile**, and finally pass the **Containerfile** to Podman which then packages and creates your automation execution environment image. The definition file we will create for Ansible Builder is in **yaml** format and contains different sections which we will discuss in further detail.

The following is an example of a definition file:

#### Example 2.1. A definition file

```
version: 1

build_arg_defaults: 1
  ANSIBLE_GALAXY_CLI_COLLECTION_OPTS: "-v"

ansible_config: 'ansible.cfg' 2

dependencies: 3
  galaxy: requirements.yml
  python: requirements.txt
  system: bindep.txt

additional_build_steps: 4
  prepend: |
    RUN whoami
    RUN cat /etc/os-release
  append:
    - RUN echo This is a post-install command!
    - RUN ls -la /etc
```

- 1 Lists default values for build arguments
- 2 Specifies the **ansible.cfg** file path
- 3 Specifies the location of various requirements files

## 4 Commands for additional custom build steps

For more information about these definition file parameters, please see [this section](#).

## 2.3. EXECUTING THE BUILD AND CREATING COMMANDS

### Prerequisites

- You have created a definition file

### Procedure

To build an automation execution environment image, run:

```
$ ansible-builder build
```

By default, Ansible Builder will look for a definition file named **execution-environment.yml** but a different file path can be specified as an argument via the **-f** flag:

```
$ ansible-builder build -f definition-file-name.yml
```

where *definition-file-name* specifies the name of your definition file.

## 2.4. BREAKDOWN OF DEFINITION FILE CONTENT

A definition file is necessary for building automation execution environments with Ansible Builder, as it specifies the content which will be included in the automation execution environment container image.

The following sections breaks down the different parts of a definition file.

### 2.4.1. Build args and base image

The **build\_arg\_defaults** section of the definition file is a dictionary whose keys can provide default values for arguments to Ansible Builder. See the following table for a list of values that can be used in **default\_build\_args**:

Value	Description
<b>ANSIBLE_GALAXY_CLI_COLLECTION_OPTS</b>	<ul style="list-style-type: none"> <li>Allows the user to pass the <b>-pre</b> flag to enable the installation of pre-releases collections</li> <li><b>-c</b> is the equivalent of setting <b>verify_ssl</b> to <b>false</b></li> </ul>
<b>EE_BASE_IMAGE</b>	Specifies the parent image for the automation execution environment, enabling a new image to be built that is based off of an already-existing image
<b>EE_BUILDER_IMAGE</b>	Specifies the image used for compiling-type tasks

Value	Description
-------	-------------

The values given inside **build\_arg\_defaults** will be hard-coded into the **Containerfile**, so these values will persist if **podman build** is called manually.



#### NOTE

If the same variable is specified in the CLI **--build-arg** flag, the CLI value will take higher precedence.

### 2.4.2. Ansible config file path

When using an **ansible.cfg** file to pass a token and other settings for a private account to an automation hub server, list the config file path (relative to where the definition file is located) as a string to enable it as a build argument in the initial phase of the build.

The **ansible.cfg** file should be formatted like the following example:

#### Example 2.2. An **ansible.cfg** file

```
[galaxy]
server_list = automation_hub

[galaxy_server.automation_hub]
url=https://cloud.redhat.com/api/automation-hub/
auth_url=https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/token
token=my_ah_token
```

For more information on how to download a collection from automation hub, please see the related Ansible documentation page.

### 2.4.3. Dependencies

#### 2.4.3.1. Galaxy

The **galaxy** entry points to a valid requirements file for the **ansible-galaxy collection install -r ...** command.

The entry **requirements.yml** may be a relative path from the directory of the automation execution environment definition's folder, or an absolute path.

The content of a **requirements.yml** file may look like the following:

#### Example 2.3. A **requirements.yml** file for Galaxy

```
---
collections:
  - geerlingguy.java
```

```
- kubernetes.core
```

### 2.4.3.2. Python

The **python** entry in the definition file points to a valid requirements file for the **pip install -r ...** command.

The entry **requirements.txt** is a file that installs extra Python requirements on top of what the Collections already list as their Python dependencies. It may be listed as a relative path from the directory of the automation execution environment definition's folder, or an absolute path. The contents of a **requirements.txt** file should be formatted like the following example, similar to the standard output from a **pip freeze** command:

#### Example 2.4. A requirements.yml file for Python

```
boto>=2.49.0
botocore>=1.12.249
pytz
python-dateutil>=2.7.0
awxkit
packaging
requests>=2.4.2
xmldict
azure-cli-core==2.11.1
python_version >= '2.7'
collection community.vmware
google-auth
openshift>=0.6.2
requests-oauthlib
openstacksdk>=0.13
ovirt-engine-sdk-python>=4.4.10
```

### 2.4.3.3. System

The **system** entry in the definition points to a **bindep** requirements file, which will install system-level dependencies that are outside of what the collections already include as their dependencies. It may be listed as a relative path from the directory of the automation execution environment definition's folder, or an absolute path.

To demonstrate this, the following is an example **bindep.txt** file that adds the **libxml2** and **subversion** packages to a container:

#### Example 2.5. A bindep.txt file

```
libxml2-devel [platform:rpm]
subversion [platform:rpm]
```

## 2.4.4. Additional custom build steps

The **prepend** and **append** commands may be specified in the **additional\_build\_steps** section. These will add commands to the **Containerfile** which will run either before or after the main build steps are executed.

The syntax for **additional\_build\_steps** must be one of the following:

- a multi-line string

#### Example 2.6. A multi-line string entry

```
RUN whoami
RUN cat /etc/os-release
```

- a list

#### Example 2.7. A list entry

```
- RUN echo This is a post-install command!
- RUN ls -la /etc
```

## 2.5. OPTIONAL BUILD COMMAND ARGUMENTS

The **-t** flag will give your automation execution environment image a specific name. For example, the following command will build an image named **my\_first\_ee\_image**:

```
$ ansible-builder build -t my_first_ee_image
```

If you have multiple definition files, you can specify which one to use by utilizing the **-f** flag:

```
$ ansible-builder build -f another-definition-file.yml -t another_ee_image
```

In [the example above](#), Ansible Builder will use the specifications provided in the file **another-definition-file.yml** instead of the default **execution-environment.yml** to build an automation execution environment image named **another\_ee\_image**.

For other specifications and flags that are possible to use with the build command, enter **ansible-builder build --help** to see a list of additional options.

## 2.6. CREATING A CONTAINERFILE WITHOUT BUILDING AN IMAGE

To create a shareable **Containerfile** without building an image from it, run:

```
$ ansible-builder create
```

## CHAPTER 3. PUBLISHING AN AUTOMATION EXECUTION ENVIRONMENT

### 3.1. PUSHING AN EXECUTION ENVIRONMENT CONTAINER IMAGE TO AUTOMATION HUB

#### Prerequisite

- You have execution environment permissions in automation hub allowing you to create new containers or push to an existing container.

A container registry is a repository for storing container images. Once you have built an automation execution environment image, you'll be ready to push that container image to the registry portion of your instance of automation hub.

With your automation hub URL on hand, run the following command to log in to Podman, substituting your username, password, and automation hub URL:

```
$ podman login -u=username -p=password automation-hub-url
```

Once you're logged in to Podman, run the following command to push your container image to the container registry on automation hub:

```
$ podman push automation-hub-url/ee-image-name
```



#### NOTE

The automation execution environment image name is specified by the **-t** argument to the **ansible-builder build** command. If you did not specify a custom image name using the **-t** flag, the default image tag is **ansible-execution-env:latest**.

### 3.2. PULLING FROM A PROTECTED REGISTRY

To pull container images from a password or token-protected registry, create a credential in automation controller:

#### Procedure

1. Navigate to automation controller
2. In the side-menu bar, click **Resources** > **Credentials**.
3. Click **Add** to create a new credential.
4. Supply an authorization URL, username, and password. Click **Save**.

For more information, please reference the Pulling from Protected Registries section of the Execution Environment documentation.

## CHAPTER 4. BUILDING OFF OF EXISTING BASE EES PROVIDED BY RED HAT ANSIBLE AUTOMATION PLATFORM

### 4.1. GATHERING SYSTEM-LEVEL DEPENDENCIES

The **bindep** format provides a way of specifying cross-platform requirements. A minimum expectation is that the collection(s) specify necessary requirements for **[platform:rpm]**.

Below is an example of content from a valid **bindep.txt** file:

#### Example 4.1. A **bindep.txt** file

```
python38-devel [platform:rpm compile]
subversion [platform:rpm]
git-lfs [platform:rpm]
```

Entries from multiple collections will be combined into a single file. This will be processed by **bindep** and then passed to **dnf**. Only requirements with no profiles or no runtime requirements will be installed to the image.

### 4.2. NOTE FOR PIP-BASED REQUIREMENTS

Python requirements files are combined into a single file using the **requirements-parser** library in order to support complex syntax. Entries from separate collections that give the same package name will be combined into the same entry, with the constraints combined.

There are several package names which are specifically ignored by **ansible-builder**; if a collection lists these, they will not be included in the combined file. These include test packages and packages that provide Ansible itself.

The full list can be found in **EXCLUDE\_REQUIREMENTS** in the [ansible\\_builder.requirements.py](#) module.

### 4.3. CUSTOMIZING AN EXISTING EXECUTION ENVIRONMENT IMAGE

Ansible Controller ships with three default execution environments:

- **Ansible 2.9** - no collections are installed other than Controller modules
- **Minimal** - contains the latest Ansible 2.11 release along with Ansible Runner, but contains no collections or other additional content
- **EE Supported** - contains all Red Hat-supported content

While these environments cover many automation use cases, you can add additional items to customize these containers for your specific needs. The following procedure adds the **kubernetes.core** collection to the **ee-minimal** default image:

#### Procedure

1. Log in to **quay.io** via Podman:

-

```
$ podman login -u="[username]" -p="[token/hash]" quay.io
```

2. Verify that you have access to pull the base image that you want to use.
  - a. For example, to pull a **ee-minimal** image, check that you have access to [this repository](#).
3. Configure your Ansible Builder files to specify any additional content to add to the new execution environment image which is based off of **ee-minimal**.
  - a. For example, to add the [Kubernetes Core Collection from Galaxy](#) to the image, fill out the **requirements.yml** file as such:

```
---
collections:
  - kubernetes.core
```

- b. For more information on definition files and their content, refer to [to definition file breakdown section](#).
4. In the execution environment definition file, specify the filepath to the original **ee-minimal** container via the **EE\_BASE\_IMAGE** field. In doing so, your final **execution-environment.yml** file will look like the following:

#### Example 4.2. A customized **execution-environment.yml** file

```
---
version: 1

build_arg_defaults:
  EE_BASE_IMAGE: 'example.registry.com/my-base-ee'

dependencies:
  galaxy: requirements.yml
```



#### NOTE

Since this example uses the community version of **kubernetes.core** and not a certified collection from automation hub, we do not need to create an **ansible.cfg** nor reference that in our definition file.

5. Build the new execution environment image using the following command:

```
$ ansible-builder build -t quay.io/[username]/new-ee
```

where **[username]** specifies your username, and **new-ee** specifies the name of your new container image.

- a. Use the **podman images** command to confirm that your new container image is indeed in that list:

#### Example 4.3. Output of a **podman images** command with the **imagenew-ee**

```
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
localhost/new-ee latest   f5509587efbb  3 minutes ago  769 MB
```





6. Verify your newly-created execution environment image via Ansible Navigator
7. Push your image to the container registry automation hub:

```
$ podman push quay.io/[username]/new-ee
```

8. Pull your new image in your automation controller instance:
  - a. Navigate to automation controller
  - b. From the side-navigational bar, click **Administration** > **Execution Environments**.
  - c. Click **Add**.
  - d. Enter the appropriate information then hit **Save** to pull in the new image.

**NOTE**

if your instance of automation hub is password or token protected, ensure that you have the appropriate container registry credential set up.